(4)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NW-LIS-89-12-04
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): Balance in Architectural Design
5. TYPE OF REPORT & PERIOD COVERED: Technical
6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Samuel Ho, Larry Snyder
8. CONTRACT OR GRANT NUMBER(s): N00014-88-K-0453
9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Northwest Laboratory for Integrated Systems
University of Washington
Dept. of Comp. Science, FR-35 Seattle, WA 98195
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS:
DARPA-ISTO
1400 Wilson Boulevard
Arlington, VA 22209
12. REPORT DATE: December 1989
13. NUMBER OF PAGES: 16
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office):
Office of Naval Research - ONR
Information Systems Program - Code 1513: CAF
800 North Quincy Street
Arlington, VA 22217
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report):

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

Architectural design, VLSI theory, price/performance ratio, VLIW processor architecture, bit serial, word-parallel, RISC, CISC.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

We introduce a performance metric, "normalized time," which is closely related to such measures as the area-time product of VLSI theory, and the price / performance ratio of advertising literature. This metric captures the idea of a piece of hardware "pulling its own weight," i.e, contributing as much to performance as it costs in resources. We then prove general theorems for stating when the size

Continued on back page....

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

#20 Abstract, (continued from front page)

of a given part is in balance with its utilization, and give specific formulas for commonly found linear and quadratic devices. We also apply these formulas to an analysis of a specific processor element, and discuss the implications for bit-serial vs. word-parallel, RISC vs. CISC, and VLIW designs.

Balance in Architectural Design

Abstract

We introduce a performance metric, *normalized time*, which is closely related to such measures as the area-time product of VLSI theory, and the price / performance ratio of advertising literature. This metric captures the idea of a piece of hardware "pulling its own weight," i.e. contributing as much to performance as it costs in resources. We then prove general theorems for stating when the size of a given part is in balance with its utilization, and give specific formulas for commonly found linear and quadratic devices. We also apply these formulas to an analysis of a specific processor element, and discuss the implications for bit-serial *vs* word-parallel, RISC *vs* CISC, and VLIW designs.

1 Introduction

Architectural design – in buildings – demands both art and engineering from the architect. Architectural design of computers also requires art and engineering, but, to date, the rationale for computer designs seems not to be founded on either a clear artistic basis, a matter which we do *not* consider further, nor a clear engineering basis, a matter which we will.

Specifically, we will conceptualize a computer as a collection of its components, e.g. a datapath, ALU, register file, etc., capable of running programs. New components, added to this base architecture, will carry a cost and, presumably, deliver a performance improvement. Using this point of view, a mathematical formulation of principles of computer design can be developed. One such principle emerges stating that optimal designs must be balanced.

> **Balance:** The cost of a given part relative to the cost of the entire system must be equal to the time on the critical path spent by that part, relative to the total running time.

The principle that computer parts have to "pull their own weight" clearly makes sense. However, by giving it mathematical precision, we are able to quantify many aspects of computer design. Thus, qualitative statements such as "CISC machines have too much control," which have been bandied about in connection with the RISC / CISC controversy, can be made quantitatively precise to "four decimal places."

The basis for our mathematical formulation is a concept of *normalized time* which we developed from a concept of *normalized analysis* [HS89] developed by Holman in his doctoral dissertation [H88].

In his thesis [1988] Holman analyzed parallel computer processor elements. He had to measure both cost and performance improvement. To expose the performance improvement due to a component, such as a floating point coprocessor, he compared machines whose PE's had the component with machines whose PE's did not. A difference in performance could then be ascribed to this component. To keep the machines on the same cost basis, he allowed the parallel machine without the feature to have more PE's until the total amount of hardware was the same in both machines. Comparing the two machines on benchmarks showed either a performance improvement for the enhanced PE's, suggesting it to be a worthwhile addition to a parallel machine, or a performance loss, showing it to be worthless. Thus, Holman's normalized analysis [HS89] compared PE enhancements with a "no action" alternative, added parallel PE's.

There were two problems with normalized analysis. First, it applied only to parallel computer processor elements, since it is necessary to equalize the costs of the two machines by adding additional parallel processors to the one without the enhancement. Second, normalized analysis applied to computer components on a "take it or leave it" basis, i.e. a PE either has a multiplier or not; considering, say, the benefit of a smaller multiplier couldn't be done, except by tedious independent analyses. But the idea of downsized components makes sense from a cost performance point of view. Holman observed: *Reducing the width of a shifter by half reduces its area by a factor of four, but cuts performance by only a factor of two* [H88].

The first problem, the limitation to parallel PE's, was solved [HHS89] by introducing the concept of *normalized time*. (See next section.) The solution of the second problem, scaling components, is a contribution of this paper. We introduce *parameterized enhancements.*

We develop a theory of parameterized enhancements and prove two fundamental theorems about normalized time for computer components: One applies to components whose cost grows linearly, such as adders, buses, register files, etc. and the second applies to quadratically growing components such as multipliers, shifters, etc. We also show how to evaluate waiting time in evaluating a design for parallel machines. Another contribution of this paper is to illustrate the theory by applying it to a specific set of components designed in $3\mu m$ CMOS. Though the examples are necessarily sensitive to the VLSI implementation, they show how the theory is applied, and suggest trends that might be more general. As an example, our numbers show that the optimal size of a VLIW machine is about 7 instructions wide, the width of the (small) Multiflow machine [M87].

The paper begins with a definition of normalized time, an analysis of waiting time, and a derivation of the Balance principle. There follows a mathematical

development for linear and quadratic components. The theorems are then illustrated in an extensive applications section that is interspersed with discussion about the relationship of the results to matters such as bit serial *vs* word parallel, RISC *vs* CISC, and optimal VLIW.

2 Normalized Model

The first order of business is to fix the environment for the analysis. All analyses will be conducted with respect to some target computation, which is fixed in advance. A standard benchmark does well in this application, but all the usual warnings about the applicability of measurements taken from a benchmark apply. Clearly, some designs are more suited to some problems than others.

We make all of our comparisons relative to a reference design known as the *base architecture*. As such, the units specified for the time T and cost C are irrelevant, since they will cancel when making any comparisons. For convenience, we set the time and cost of the base architecture T_0 and C_0 to be unity.

The *normalized time* of a program running on a specific machine is the product of its time and cost, TC, when running on that machine. Normalized time allows trading time for cost, and vice versa. So, for example, a parallel machine whose processor elements are half as fast, but cost half as much as another machine's PE's can compensate by having twice as many. Or, replacing an ALU with one that costs twice as much but runs twice as fast won't change the normalized time of a program. In each case, there is an assumption that the change in hardware can be used: Adding processor elements assumes there is sufficient parallelism in the program; accelerating the ALU assumes operations get done faster and are not inhibited by other factors.

Such assumptions are often reasonable, at least over small ranges. As with any theory, however, the extent to which the assumptions are realized must be assessed when interpreting the results.

To the base architecture we will add an enhancement, such as a multiplier, floating point unit, or the like. By analogy to Amdahl's law [A67], we assume that some fraction f of the total computation time is affected by the enhancement, and the rest is left undisturbed. This fraction would be the fraction of time spent multiplying for multipliers, and so forth. We also assume that the affected part of the computation speeds up by a constant factor S, and that the improvement carries a cost c.

Definition 1 *An enhancement affecting a fraction f with speedup S and cost c results in time*

$$T = 1 - f + \frac{f}{S}$$

and cost

$$C = 1 + c.$$

At this point, we generalize from a single enhancement to a range of enhancements. This is the simple expedient of adding a parameter u which parameterizes the value of the speedup $S(u)$ and cost $c(u)$. We assume that the affected fraction f remains constant, since the family of enhancements should be closely enough related that they affect the same operations.

For many types of enhancements, the family of potential enhancements is discrete, but still large. An example is the varying widths of multiplier or shifter elements. In these cases, we may approximate the discrete parameter by a continuous one, bearing in mind the possibility of intermediate results which lie between values of the discrete parameter.

In addition to this categorization of the total time into the parts affected or unaffected by the enhancement, there is another important effect, especially for parallel systems. In a parallel computer, a processing element may often find itself without anything to process, because it is waiting for something to happen.

Let us assume that the base architecture spends a fraction W_0 of its time waiting. We also denote the waiting time in the modified architecture by W. Then, the modified running time is given by

$$T = 1 - W_0 - f + W + \frac{f}{S}.$$

The waiting time can, in principle be an arbitrarily complicated function. However, a linear function is a reasonable approximation. Such a function contains two major terms. The constant term ω_0 represents waiting that does not depend on the speed of the rest of the computation, such as communication delay. The linear term ωT is proportional to the speed of the computation, and encapsulates waiting associated with acquiring results from other subcomputations. This results in waiting time

$$W = \omega_0 + \omega T.$$

Substituting this into our original formula gives

$$T = 1 - \omega_0 - \omega - f + \omega_0 + \omega T + \frac{f}{S}.$$

From this, bringing terms dependent on T to the left yields

$$(1 - \omega)T = 1 - \omega - f - \frac{f}{S},$$

which is

$$T = 1 - \frac{f}{1 - \omega} + \frac{f}{(1 - \omega)S}.$$

This is equal to

$$T = 1 - f' + \frac{f'}{S}$$

if we define an adjusted fraction

$$f' = \frac{f}{1-\omega}.$$

Thus, the constant term of waiting is indistinguishable from computation that is not affected by the enhancement, and the linear term is equivalent to an adjustment in the fraction affected by the enhancement. In the remainder of the discussion, we make no further mention of waiting time, but include it implicitly, by the adjustments described above.

3 A General Principle

Now, let us recall that we have described the time and cost of an enhancement by using a parameter. The first theorem expresses the conditions for which the normalized time is at a local minimum. This occurs when the fraction of cost added by a further step is equal to the fraction of time removed by that step. At this point, small changes in the size of the enhancement exactly balance small changes in its cost. In general, though, there may be many such points, which may be maxima, minima, or points of inflection. Furthermore, if the range of the parameter is restricted, the absolute minimum may lie at an end point, instead of a local minimum.

Theorem 1 *Given the cost $C(u)$ and time $T(u)$, parameterized by u, the normalized time is minimized at a point where*

$$\frac{C'(u)}{C(u)} = -\frac{T'(u)}{T(u)}.$$

The minimum occurs when the first derivative of the normalized time is zero, or the equation

$$0 = \frac{d}{du}(TC) = T'(u)C(u) + T(u)C'(u).$$

Rearranging this equation provides the desired result. □

This theorem provides a mathematical justification, and a quantitative test, for the concept of a machine component pulling its own weight. Therefore, we can state a general principle for choosing the size of an enhancement.

> **General Principle of Design:** To be cost-effective, the additional cost of an enhancement, relative to the cost of the whole, must be outweighed by the reduction in computation time resulting from the enhancement, relative to the total time.

4 Common Solutions

Substituting several common functions for the time and cost functions permits solving for the optimum size of enhancement.

4.1 Linear Speedup

Linear scalability characterizes a large number of circuits. In such circuits, a circuit of twice the size performs a given operation in half the time. Any design limited by register transfers or communication will be linearly scalable. For instance, an k-bit wide adder will require n/k cycles to perform an n-bit addition. A bus of k wires requires n/k cycles to transmit an n-bit number.

Definition 2 *A linear speedup has time given by* $T(u) = 1 - f + f/u$ *and cost given by* $C(u) = 1 - \alpha + \alpha u$, *where* α *and* f *are constants between* 0 *and* 1.

The constants α and f represent the fraction of the cost and time, respectively, associated with the enhancement. The remainder of the cost and time is not affected by the enhancement.

Theorem 2 *The width at which the normalized time of a linear speedup is optimal is given by the formula*

$$u = \sqrt{\frac{f(1-\alpha)}{\alpha(1-f)}}.$$

For a linear speedup, the normalized time is

$$C(u)T(u) = 1 - \alpha + \alpha u - f + \frac{f}{u} + 2\alpha f - \alpha f u - \frac{\alpha f}{u}.$$

Setting the first derivative to zero results in the equation

$$0 = \frac{d}{du}(CT) = \alpha - \frac{f}{u^2} - \alpha f + \frac{\alpha f}{u^2}$$

This produces the quadratic equation

$$\alpha(1-f)u^2 - f(1-\alpha) = 0$$

which has the solution

$$u = \sqrt{\frac{f(1-\alpha)}{\alpha(1-f)}}$$

as in the theorem.

Verification that this is indeed a minimum comes from examining the sign of the second derivative, which is

$$\frac{d^2}{du^2}(CT) = \frac{2f}{u^3} - \frac{2\alpha f}{u^3} = \frac{2f(1-\alpha)}{u^3} > 0. \ \square$$

A corollary to this theorem is that the processor is already balanced with respect to a linear enhancement when the time and cost fractions are equal. If these fractions are equal, a small change to the size of the enhancement has no effect, so there is no need for any departure from the base architecture. Furthermore, since linear enhancements have normalized time concave upward, there is no other lower minimum.

Corollary 3 *For a linear enhancement, the base architecture is already optimal in the case when* $\alpha = f$.

Substituting $\alpha = f$ means that the optimal value is at $u = 1$, or the original architecture. $\square$

4.2 Quadratic Growth

A second major class of enhancements is that of the quadratic circuits. The transitive functions, including multiplication, cyclic shifting, and other problems, exhibit an area-time tradeoff of the form AT^2 in the VLSI model [T79, V80]. We shall adapt that argument to find the size which optimizes the normalized time.

The original argument considers the minimum bisection width w. In any transitive function, there exists some input such that $\Omega(n^2)$ bits must cross this bisection, which requires time $\Omega(n^2/w)$. On the other hand, since any bisection cuts at least w wires, the total wire area is $\Omega(w^2)$. This results in the requirement of $AT^2 = \Omega(n^2)$.

We use w directly, to parameterize the various circuits. We assume that for each w, there is a corresponding circuit for the function requiring time $O(n^2/w)$ and area $O(w^2)$. We also have, as usual, a fraction f of transitive function computation in the circuit, and a fraction α of the total cost charged to the transitive function circuit. The factor of n^2 may be absorbed into these constants, as well.

A transitive function computation obeys the conditions

$$\begin{aligned} T(w) &= 1 - f + \frac{f}{w} \\ C(w) &= 1 - \alpha + \alpha w^2 \end{aligned}$$

with the restrictions $0 < f < 1$, $0 < \alpha < 1$, and $w > 0$, to avoid singularities.

To find the minimum, we set the derivative to zero, producing the equation

$$0 = \frac{\partial}{\partial w} TC = \frac{1}{w^2}\left(2\alpha(1-f)w^3 + \alpha f w^2 - f(1-\alpha)\right).$$

Since both α and f are between 0 and 1, this derivative is negative near $w = 0$, and positive at large w. Therefore, there exists at least one point where it is zero, since the derivative is continuous.

In order to obtain a numerical solution, we multiply the derivative by w^2, to get the cubic equation

$$2\alpha(1-f)w^3 + \alpha f w^2 - f(1-\alpha) = 0.$$

The final numerical value may be obtained from the cubic equation formula for equations $w^3 + pw^2 + r$ by setting the values

$$\begin{aligned}
p &= \frac{f}{2(1-f)} \\
r &= \frac{f(1-\alpha)}{2\alpha(1-f)} \\
a &= -\frac{1}{12}\left(\frac{f}{1-f}\right)^2 \\
b &= \frac{1}{108}\left(\frac{f}{1-f}\right)^3 - \frac{f(1-\alpha)}{2\alpha(1-f)} \\
A &= \sqrt[3]{-\frac{b}{2} + \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}} \\
B &= \sqrt[3]{-\frac{b}{2} - \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}} \\
x &= A + B \\
w &= x - \frac{p}{3}. \quad \square
\end{aligned}$$

5 Applications of the Analysis

We now turn to several examples of normalized analysis. The first example is setting up the equations for the quadratic functions of shifting and multiplication. Then, we consider concrete values for scaling a 32-bit processor, first downward to smaller word sizes, then upward to a VLIW structure. We can also combine these last two into a single, piecewise linear analysis, covering an entire span of datapath sizes.

5.1 Quadratic Functions

Here, we make a more careful study of shifters and multipliers. The AT^2 results provide only a lower bound, not a tight bound. Furthermore, even if the AT^2 tradeoff holds, the lower order terms neglected in the computation can be larger than the highest order term at the moderate widths seen in actual design.

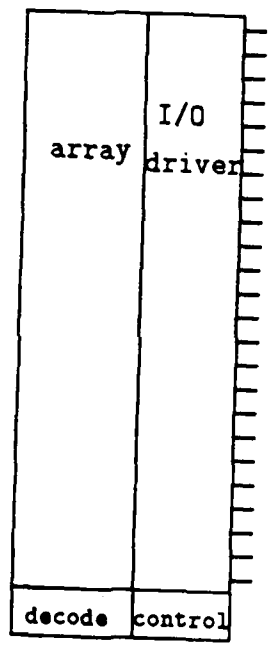

Figure 1: The Quarter Horse shifter

For instance, we can take the shifter from the Quarter Horse processor [HJK-STY85], depicted in Figure 1. This is a 32-bit crossbar shifter, of approximately two million square microns, in the $3\mu m$ CMOS process. If we examine the actual design, though, we see that the crossbar array is only 42 percent of the width of the shifter. The remaining 58 percent consists of the input latches, the array input and output cells, and the output drivers. In the vertical dimension, the 32 bit-slices make up 90 percent of the shifter height, with 10 percent devoted to decoders and control logic. Thus, if we set w to be the width, relative to the 32-bit base architecture, we have cost $(0.58 + 0.42w)(0.1 + 0.9w)$. Thus, the cost is $0.38w^2 + 0.56w + 0.06$. With this cost, the linear term is larger than the quadratic term until the shifter reaches at least 47 bits. This means that to test Holman's observations concerning narrow shifters, the linear model is most accurate for 32 bit words.

This cost function, though is highly sensitive to the VLSI implementation. The shifter from the RISC II microprocessor [KSPS83] is quite similar, but is designed in a $4\mu m$ nMOS process. This shifter, though, has a total area of $640,700\lambda^2$. This area is divided into $417,200\lambda^2$ for the crossbar array, and $223,500\lambda^2$ for the input latches, drivers, and decoders. The cost function, omitting the relatively small control area, is then $0.65w^2 + 0.35w$. The balance of the linear and quadratic terms has been reversed, compared with the Quarter Horse shifter. In fact, the quadratic term outweighs the linear term for any size larger than 17 bits.

Another case is that of the multiplier. Here, while the AT^2 results require a lower bound of n^2/w for the running time, given a width w, the straightforward method of accumulating partial products on w bits of the operands produces a running time of n^2/w^2. If we change our function for the running time accordingly, to $1 - f + f/w^2$, the equation becomes identical to that for a linear enhancement, except that w has been replaced by w^2. As a result, the optimal width is now given by

$$w = \sqrt{\frac{f(1-\alpha)}{\alpha(1-f)}}.$$

Both of these cases show how we can adjust the theoretical results given previously to account in a more detailed way for the actual behavior of a datapath element. The modified functions can be directly substituted, and solving these revised equations will provide a better estimate of the optimal size.

5.2 Datapath Width – Smaller Processors

Next, we can consider changing the width of the entire datapath, taken as a whole. The extremes of this experiment are the bit-serial and word-parallel processors, but there are several intermediate widths, also. Clearly, the parameter must be the width, in bits, of the datapath.

Component	Area
Register File (16)	445,603 λ^2
ALU	404,954 λ^2
Shifter	424,844 λ^2
Control	466,041 λ^2
Communication (8)	1,458,000 λ^2
Total	3,199,442 λ^2

Table 1: Processor element area estimates.

Once we make this choice, we must now divide the time and area of a processor and program into the affected and unaffected categories. For this example, we provide some concrete values for the parameters. The analysis below is based on estimates [HS89] of sizes of elements taken from the Quarter Horse [HJK-STY85] and Mosaic [LRSS84] processor elements, scaled to a $3\mu m$ process, for a hypothetical processor element. See Table 1.

The category of area that is not affected by a change in the datapath width can be described, loosely, as control and storage. This category includes the actual control, in the form of random logic, microcode, and the like, as well as support structures, such as the program counter. Storage is also unaffected, since changing the storage capacity is a separate question. (We may also study this question, with structures such as the register windows found in the Berkeley RISC [PS82]. Here, though, the parameter would be the number of registers, and not the width of the datapath.)

The parts that are affected by the width include the ALU, as well as any shifter or multiplier that may be present. Also, many other, less prominent, parts are also affected. For instance, drivers and latches for bus interfaces and register bit lines are affected by datapath width. Though these buffers are smaller than large parts the size of an ALU, there is typically one bus driver and one latch for each datapath element, so their cumulative effect is significant.

Next, we determine the sizes for a specific processor. As we have postulated above, the registers, control, and communication are unaffected by changes to the datapath width. Furthermore, we postulate that the ALU and shifter are affected linearly. While one might expect the shifter array to be quadratic, its buffers and bus drivers are linear, and, as we noted earlier, the buffers are actually larger than the shifter array. This effect becomes even more pronounced as we move to smaller sizes. This results in a total unaffected area of $2,369,644\ \lambda^2$, and a total linearly affected area of $829,798\ \lambda^2$. The ratios are then 0.74 unaffected to 0.26 affected. If we specify the cost in the standard form of $1-\alpha+\alpha w$, we have $\alpha = 0.26$, and w defined as the datapath width divided by 32.

Next the value for f is obtained for several benchmarks [HS89]. Since the original values are for f and W, we must adjust for the waiting time to give an

Program	f	W	f'	w
Bitonic Sort	.75	.17	.90	162
Matrix Product	.94	.03	.97	307
LU Decomposition	.76	.17	.92	183
Cholesky Decomp.	.25	.73	.89	154
Jacobi Method	.91	.04	.95	235
SOR Method	.96	.02	.98	377
SIMPLE	.71	.27	.97	307

Table 2: Fraction of time affected by data path width.

effective fraction f'. These are shown in Table 2.

We can then use the formula from the linear analysis to find the best data path width. This is given by

$$32\sqrt{\frac{f(1-\alpha)}{\alpha(1-f)}}.$$

The results are also given in Table 2.

The datapath widths shown in Table 2 are enormous, and require some careful examination. First, even before considering the numbers, observe that enormous numbers are to be expected based on the values of $\alpha = 0.26$ and the f' values of about 0.9 given in Table 2. An architecture is in balance when $\alpha = f'$, so one fully expects the equations to mandate a dramatic increase in width when the gap between α and f' is so large.

Second, as explained earlier (Section 5.1), the linear formulation of the shifter is accurate only to about 47 bits; after that, a quadratic formulation is needed. Beyond that, the nature of the computation changes after 32 or 64 bits. Discussion of datapath width concerns operations on values within an instruction. Most programs have little need for integers suitable for representing the number of electrons in the universe. When the widths get as large as those shown in Table 2, we have moved into a VLIW situation – separate instructions operating on subfields of the datapath. (See Section 5.3 below.) This is simply a different case, with different growth. The best way to interpret this analysis is that *normalized time for a datapath diminishes throughout the range of applicability.* This confirms Holman's [1988] analysis (in the range 1 to 32, a datapath of 32 is best) conjectured on the basis of independent tests of a few widths in this range.

Third, even with an interest in BIGNUMs and a proper mathematical model, the data are not quite precise enough to make an exact prediction of datapath speedup. The values for f in Table 2 are the fraction of all instructions using the datapath, but what we really need is the fraction of the time the datapath

is in use. These two numbers would be the same if all instructions took the same amount of time and all overhead (fetching, decoding, etc.) were hidden, say in a pipeline. (Of course, getting data of this kind is difficult, but now that we know what we need, future experiments can be more useful.) So, the proper interpretation of w, as found in Table 2, is "the datapath should be at least 32 bits wide."

We now turn to the more general question of bit-serial and word-parallel computers. Given the high fraction of computation affected by datapath width, the analysis essentially states that control logic is dead weight. As such, it should be amortized over as large a number of datapath bits as possible. Using a word-parallel processor meets this requirement. On the other hand, a bit-serial processor seems to be the worst possible arrangement, with only one bit of data path per unit of control.

Using SIMD design for bit serial processors solves this problem, of course. A large part of the fixed cost logic, that dealing with instruction fetch, decoding, and control flow, is amortized over not one, but 65,536 bits of data path, in the Connection Machine, for example. Effectively, these parts are now scaled by neither datapath width nor number of processors, so they disappear from both sides of the equation. Thus, an SIMD architecture can help greatly in restoring balance in a processor.

We can also consider this problem from the viewpoint of instruction set complexity. The RISC *vs* CISC controversy is not as closely related to the results of this analysis, but we can describe some of the principle arguments as attempts to solve the problem of datapath balance.

On the RISC side of the coin, we have seen how most processors have, relatively, too much control for a given amount of data path. The RISC advocates address this by reducing the amount of control. This reduces the unaffected cost term in the normalized time analysis. (Using the extra space for another part, such as the register windows of the Berkeley RISC [PS82] is actually two separate changes, first deleting the extra control, then adding the register windows, which should be analyzed separately.)

On the CISC side, the argument is that the problem is not so much an excess of control, but an inability to use a larger data path. Thus, the solution is to add some more control, and larger instructions, so that there is more chance to exploit parallelism within an instruction. Having several words in process simultaneously then effectively allows for a wider data path.

5.3 Wider Processors – VLIW

Even among word-parallel designs, the desired data path widths are still much larger than the average word size. The VLIW design addresses this concern. By putting several operations into one instruction word, the effective word size is larger, allowing use of wider data paths. Meanwhile, the amount of control is increased by only a small amount.

In a VLIW design, though, it is often the case that, because of data or control dependencies, the extra calculations possible are wasted. Our model can take this into account, by repeating the analysis, but with a value of f reflecting, now, the fraction of time that two or more operations are simultaneously ready to execute. In fact, we can generalize f to a piecewise linear function, and still carry out the analysis. The data of how many instructions are ready to be issued at each time step are not exactly known, but the data of Table 2 are for a 64-way parallel computation, so generally, there would be a substantial number.

In our analysis of VLIW processors, we will use the same processor as the last analysis. We will make a large number of simplifying assumptions, but the results should still provide at least a general guideline. We assume that the register file now grows linearly, since each ALU segment needs its own temporary storage. The ALU, of course, still grows linearly. The shifter is now a linear, and not quadratic, part, since there is no need to shift data all the way across the processor, but only within one set of operands. As a result, the width is fixed at 32 columns, but the height grows with the number of bits of datapath. We assume that the control and communication sections are affected to only a negligible degree by the changes needed for the VLIW structure.

These assumptions result in an affected area of $1,275,401\lambda^2$, and an unaffected area of $1,924,041\lambda^2$. When normalized into the form $1 - \alpha + \alpha w$, this is a value of $\alpha = 0.40$. The parameter w is taken as the size, in words, of the datapath.

We next assume, in the absence of data to the contrary, that the fraction of computation f affected by the extra ALU capacity is equal to the fraction of computation affected by reducing the datapath. In defense of this assumption, note that the benchmarks are already designed for highly parallel systems, and so using a VLIW structure is akin to increasing the parallelism. We should note though, that especially for the larger word sizes, we will need to adjust the analysis again, to take into account the reduced utilization of highly parallel systems.

To solve for the optimal number of ALU's in the system, we use our familiar equation

$$w = \sqrt{\frac{f(1-\alpha)}{\alpha(1-f)}}.$$

The results appear in Table 3.

6 Conclusion

Using the model of *normalized time*, we have created a combined value that describes both the time and the cost of a processor. In this way, we can evaluate whether a particular enhancement is cost-effective. By parameterizing a set of possible enhancements, we can evaluate a range of designs in a single analysis.

Program	w
Bitonic Sort	3.67
Matrix Product	6.96
LU Decomposition	4.15
Cholesky Decomp.	3.48
Jacobi Method	5.33
SOR Method	8.57
SIMPLE	6.96

Table 3: Optimal width of VLIW datapath (in words.)

Even for the most general case, we show that an enhancement is optimized when it is "doing its fair share." We can also state quantitatively the meaning of that intuitive concept.

For more specific cases, we can provide a formula for the optimal size of a part. After supplying some experimentally determined constants, an optimal value emerges. The principal impediment to further investigation is the lack of empirical data. In one sense, this paper is a call to the experimentalists to gather more data, and apply the formulae.

In the long-standing RISC *vs* CISC and bit-serial *vs* word-parallel controversies, the model quantifies the reasoning behind the arguments for each case, and allows isolation of the aspects of a given processor or problem responsible for observed performance.

7 References

A67 G. M. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in *Proc AFIPS Vol. 30*, pp. 483-465, 1967.

H85 W. D. Hillis, *The Connection Machine*, PhD Thesis, MIT 1985.

H88 T. J. Holman, *Processor Element Architecture for Non-shared Memory Parallel Computers*, PhD Thesis, University of Washington, 1988.

HS89 T. J. Holman and L. Snyder, "Architectural Tradeoffs in Parallel Computer Design," in *Decennial Caltech Conference on VLSI*, 1989.

HHS89 S. Ho, T. Holman, L. Snyder, "Normalized Time and Its Use in Architectural Design," in *27th Allerton Conferences on Communication, Control and Computing*, 1989.

HJKSSTY85 S. Ho, B. Jinks, T. Knight, J. Schaad, L. Snyder, A. Tyagi, C. Yang, "The Quarter Horse: A Case Study in Rapid Prototyping of a 32-bit Microprocessor Chip," in *Proceedings IEEE International Conference on Computer Design: VLSI in Computers*, pp. 261-266. IEEE, October 1985.

KSPS83 M. Katevenis, R. Sherburne Jr., D. Patterson, Carlo Séquin, "The RISC II Micro-Architecture," in *VLSI '83*, F. Anceau and E. J. Aas (eds.) Elsevier Science Publishers B. V. (North-Holland), 1983.

LRSS84 C. Lutz, S. Rabin, C. Seitz, D. Speck. "Design of the Mosaic Element," in *Conference on Advanced Research in VLSI*, Artech Books, 1984.

M87 Multiflow, *Technical Summary*, Multiflow Computer Incorporated, 1987.

PS82 D. A. Patterson and C. H. Sequin, "A VLSI RISC," *Computer*, 15(9):8-21, 1982.

T79 C. D. Thompson, "Area-Time Complexity for VLSI," in *11th annual ACM Symposium on Theory of Computing*, pp. 81-88, 1979.

V80 J. Vuillemin, "A Combinatorial Limit to the Computing Power of VLSI circuits," in *21st IEEE Symposium on Foundations of Computer Science*, 1980.